Assignment 1, CS444 Christopher Mendez, Joshua Bowen

1 Command Line Commands

1.1 In First Terminal

- git clone –branch v3.14.26 git://git.yoctoproject.org/linux-yocto-3.14
- source /scratch/opt/environment-setup-i586-poky-linux.csh
- qemu-system-i386 -gdb tcp::???? -S -nographic -kernel bzImage-qemux86.bin -drive file=core-image-lsb-sdk-qemux86.ext3,if=virtio -enable-kvm -net none -usb -localtime –no-reboot –append ”root=/dev/vda rw console=ttyS0 debug”
- Make -j4 all

1.2 In Second Terminal

- gdb
- file vmlinux
- target remote :5520
- make linux –menuconfig

2 Concurrency Write-Up

Our concurrency solution involved using threads with mutexs for synchronization.

```
pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;
```

When the program is started, it creates the specified number of consumer threads using a loop. Each of these consumers when created is given a value between 0-31 and checks that element in the array in a loop until it has been filled with a number that is between 2 and 9.

```
        long j = 0;
        for(long i = 0; i < atoi(argv[1]); ++i){
                pthread_create(&(threads[i]),
                               NULL,
                               consumer,
                               (void *)( &a[j]));
                j++;
                if(i==32){
                        j = 0;
                }
        }
```

The consumer then grabs locks on the mutex and modifies the buffer, removing the values at that element. It then unlocks and sleeps for the specified time, then prints the values.

```
void *consumer(void *tid)
{
        struct args *a = (struct args*)tid;
        while(1){
                if (a->sleep_time<10 && a->sleep_time>1 && a->tid > -1 && a->tid <32){
                        pthread_mutex_lock (&mymutex);
                        long conSleepTime = a->sleep_time;
                        long conTid = a->tid;
                        a->sleep_time = -1;
                        a->tid = -1;
                        pthread_mutex_unlock (&mymutex);
                        sleep(conSleepTime);
                return (void *)printf("Hello from thread %ld! I did %ld units of work!\
                }
        }

}
```

Meanwhile the producer(which is just a loop in the main function not a separate thread) goes between 0-31 (and if there are more consumers specified it starts back at 0) and takes the mutex locks, and fills in one of the elements in the buffer. After that it releases its locks and sleeps for 3-7 seconds before printing what it just did and repeating.

```
        for(long i = 0; i < atoi(argv[1]); ++i){
                pthread_mutex_lock (&mymutex);
                a[i].tid = i;
                long consumerSleep = genrand_int32() % 8;
                a[i].sleep_time = consumerSleep+2;

                pthread_mutex_unlock(&mymutex);
                if(mt == 1){
                        long sleepTime = genrand_int32()%5;
                        sleep(sleepTime+2);
                printf("Producer just slept for: %ld and filled tid %ld\n", sleepTime+2
                }
                else{
                        long sleepTime = 1;//rdrand code here
                        sleep(sleepTime+2);
                printf("Producer just slept for: %ld and filled tid %ld\n", sleepTime+2
                }
                if(i==31){
                        i = 0;
                }
        }
```

3 Qemu Flags

- gdb tcp::55020

open a gdbserver on TCP port 55020

- S

Do not start CPU at startup (you must type c in the monitor)

- nographic

Normally, QEMU uses SDL to display the VGA output. With this option, you can totally disable graphical output so that QEMU is a simple command line application. The emulated serial port is redirected on the console and muxed with the monitor (unless redirected elsewhere explicitly). Therefore, you can still use QEMU to debug a Linux kernel with a serial console.

- kernel bzImage-qemux86.bin

Use bzImage as kernel image

- drive file=core-image-lsb-sdk-qemux86.ext3,

A way to boot other than from a CD-ROM

- if=virtio

Check for virtio driver. If this is there we can boot KVM

- enable-kvm

Enable KVM full virtualization support. This option is only available if KVM support is enabled when compiling.

- net none

Indicate that no network devices should be configured. It is used to override the default configuration (-net nic -net user) which is activated if no -net options are provided.

- usb

Add the USB device

- localtime

Use local time

- no-reboot

Exit instead of rebooting.

- append ”root=/dev/vda rw console=ttyS0 debug”

Give the kernel command line arguments

4 Questions

4.1 What do you think the main point of this assignment is?

Set us up for success in future assignments. This assignment toured a lot of class resources, using tophat, the qemu documentation and the course webpage for tutorials and templates; using Kevin to answer any wording questions; learning git, LaTeX and how to start the kernel, and ensuring we could start and run some c code on os-class. Instead of having students meltdown and panic on harder assignments because they didnt understand how to make use of these tools, this hopefully gets some of the learning curve out of the way.

4.2 How did you personally approach the problem? Design decisions, algorithm, etc.

Upon learning the assignment was to use pthreads I figured mutexes were the way to go because they are straightforward, would meet the synchronization constraints and I am familiar with them from CS344. The way I decided to use mutexes was to lock anytime I was going to modify the shared array, store any values I needed locally, and then unlock. I didnt look for the times I would check to see what values were in the array as I didnt see this as a danger as it didnt involve modifying the memory. A decision made poorly early on was to try to make the producers and consumers their own threads. It took about an hour and a half of doing this to realize that by having a producer thread for each consumer I was making things far more complicated than need be. Instead having a for loop to generate the values works just and was simpler to implement. After I fixed this everything else fell into place.

4.3 How did you ensure your solution was correct? Testing details, for instance.

To see if the consumer/producers first I disabled the consumer sleep statement, so they wouldnt sleep at all. Then I gave the producers a sleep time of 5, and ran a stopwatch to see if the program took 25 seconds to finish with 5 consumers. Then reversed it so the producers produced without sleeping but the consumers had 5 second sleeptime. This told me that the producers and consumers didnt step over themselves and that the locks worked. To ensure they didnt step over each other I tried some other values. For example if both producers and consumers sleep for 3 seconds, then they should print at almost the same time, whereas if they are staggered so that producers sleep for 3 seconds and consumers for 6, than 2 items should be produced in the time one is consumed. All of this worked as expected brought back the randomized statements and tested for a variety of numbers, up to 50.

4.4 What did you learn?

Shallow knowledge about git, qemu and LaTeX. How to get help from Kevin, tophat and the course webpage. Refreshed on C, pthreads and synchronization.

5 Git History

Commit	Message
commit 0107728d1077ed6a170639b50df4e6f4957ecb31 Author: bowenjos ¡bowenjos@oregonstate.edu¿ Date: Sun Oct 9 22:38:17 2016 -0700	Yocto Files for real this time
commit cea3cb9b6a16439e04d0e9ac819ce772572604bf Author: bowenjos ¡bowenjos@oregonstate.edu¿ Date: Sun Oct 9 22:31:38 2016 -0700	Yocto Files 2.0
commit 2ece0dbd261eeb9954157e887db67f0f582d5d44 Merge: bb81365 cf3558b Author: bowenjos ¡bowenjos@oregonstate.edu¿ Date: Sun Oct 9 22:29:43 2016 -0700	Yocto Files
commit bb81365122a69088d250457b8e8619dad651a9ff Author: bowenjos ¡bowenjos@oregonstate.edu¿ Date: Sun Oct 9 22:09:14 2016 -0700	:...skipping...
commit 0107728d1077ed6a170639b50df4e6f4957ecb31 Author: bowenjos ¡bowenjos@oregonstate.edu¿ Date: Sun Oct 9 22:38:17 2016 -0700	Yocto Files for real this time
commit cea3cb9b6a16439e04d0e9ac819ce772572604bf Author: bowenjos ¡bowenjos@oregonstate.edu¿ Date: Sun Oct 9 22:31:38 2016 -0700	Yocto Files 2.0
commit 2ece0dbd261eeb9954157e887db67f0f582d5d44 Merge: bb81365 cf3558b Author: bowenjos ¡bowenjos@oregonstate.edu¿ Date: Sun Oct 9 22:29:43 2016 -0700	Yocto Files
commit bb81365122a69088d250457b8e8619dad651a9ff Author: bowenjos ¡bowenjos@oregonstate.edu¿ Date: Sun Oct 9 22:09:14 2016 -0700	Yocto Files
commit cf3558b7a95a5e2e83e1199c44b618f25f3c569e Author: Mendezc1 ¡mendezc@oregonstate.edu¿ Date: Sun Oct 9 13:54:28 2016 -0700	Randomization fixed to be seeded each time differently
commit d6ae8f22080239c8822f358bfe3078b418438bf9 Author: Mendezc1 ¡mendezc@oregonstate.edu¿ Date: Sat Oct 8 16:00:47 2016 -0700	Fixed some synchonization bugs, did some testing and seems to work
commit d5a4e2575a30a0ede00ee5f9d7d73abdba048745 Author: Mendezc1 ¡mendezc@oregonstate.edu¿ Date: Sat Oct 8 12:45:33 2016 -0700	Producer and consumer look to be in sync
commit 5069a6e6078d096d6b8940841cc11469c6c25bdf Author: Mendezc1 ¡mendezc@oregonstate.edu¿ Date: Sat Oct 8 12:22:32 2016 -0700	Producer and consumer thread added, still working on synchronization
commit dba844c714e8dc35ab426aeb3385979ff0c5757d Author: Mendezc1 ¡mendezc@oregonstate.edu¿ Date: Sat Oct 8 10:49:02 2016 -0700	working hello threads program
commit 56fde513c508d4eff10114eae119fff6ce69f284 Author: Christopher Mendez ¡mendezc@os-class.engr.oregonstate.edu¿ Date: Thu Oct 6 09:36:14 2016 -0700	Merging submodule
commit d176e591a3fef902715de72d8ac4dc20ae297a87 Author: Christopher Mendez ¡mendezc@os-class.engr.oregonstate.edu¿ Date: Thu Oct 6 09:19:29 2016 -0700	Added linux yocto v3.14.26 tag
commit 75b41d8d8be20fdfa5ede595c4b88e39209b8df7 Author: Christopher Mendez ¡mendezc@os-class.engr.oregonstate.edu¿ Date: Thu Oct 6 08:56:28 2016 -0700	first commit

6 Work Log

Date	Time	What
10/1/2016	10-12	Started assignment, looked into qemu emulation and threading, got stuck on configuring kernel and the producer/consumer relationship. Github created
10/6/2016	7:30-8:30, 10-12	Got questions answered by Kevin, started commiting to github, however ended up with an empty directory. Started programming thread program but didnt commit because wouldnt compile
10/7/2016	3-4	Continued working on thread program, got it to compile by itself but couldnt link it to mt19937ar.c so randomization didnt work
10/8/2016	10-1, 2-3	Gave up on linking files, so just ended up copy/pasting important parts of mt19937ar.c into concurrency.c. Looked on course website and found files that helped a lot with the threading assignment. Implemented concurrency solution, tested it and it seemed to work except the randomization wasnt being random.
10/9/2016	10-11, 1-3	Fixed randomization problem, looked into rdrand but no progress made on it. Began writeup.